Performance Implications by the Hierarchical Design of Clusters

Wissem Ben Fraj, Mohamed Essaïdi, Jens Gustedt

Performance Implications by the Hierarchical Design of Clusters*

Wissem Ben Fradj and Mohamed Essaïdi and Jens Gustedt
INRIA Lorraine & LORIA
France
email: {benfraj|Mohamed.Essaidi|Jens.Gustedt}@loria.fr

Abstract

We present experimental results for the evaluation of PC clusters that differ on several aspect of their architecture, such as being mono or biprocessors and having a high bandwidth interconnection network or not. These experiments confirm that a good equilibrium between the speed (throughput) of the different components is crucial for a satisfactory performance of such a machine. On the other hand they also show that the latency of the underlying infrastructure is of less importance and can be hidden by applying techniques from coarse grained parallelism.

1 Introduction and Overview

PC clusters that are composed of standard components (processor, bus, memory, network) are an attractive alternative to expensive multiprocessor mainframes. Unfortunately, the gap between the speed of the available processors (some GHz) and the throughput of the other components (some 10 – 100 MHz) is constantly increasing. Since several years, high performance networks are used to close this gap. These high performance components are relatively expensive, and so to obtain a better performance/cost ratio of high performance clusters a common strategy is to group several (usually two) processors into one node. By that one divides the number of (expensive) network cards by two. The implications for the performance of such a cluster are not obvious at all, since there are two contradictory effects: (1) processors share bus and network card and so their network throughput may be reduced compared to a mono-processor design with the same type of network card (2) part of the communication doesn't pass through the network (namely between processors on the same node) and can be handled in shared memory.

To evaluate the influence of such an architecture we present experiments with SSCRAP. SSCRAP is an environment for the development of coarse grained parallel applications supporting several parallel and distributed architectures. SSCRAP can use different supports for communication: of relevance in the context of clusters are MPI and PM^2. Especially the later is designed to efficiently support communication via shared memory *and* network simultaneously.

We carry out two different types of comparisons: (1) between the two clusters to discuss the advantage (or disadvantage) of a such an involved cluster design and (2) between the two different network interfaces of a biprocessor cluster. We also planed to compare the communication interfaces MPI and PM^2 to measure the possible gain provided by sharing the memory between pairs of processors but unfortunately we were not able to get the PM^2 interface functional on the high performance network interface.

This paper is organized as follows. The following section describes the environment that we used for our tests (programming support, communication libraries, hardware, OS, algorithm test set). Thereafter we present the different experimental results and conclude with a discussion on the value of different hard and software combinations.

*Part of this work was supported by the "Pôle régional lorrain *Intelligence Logicille*"

2 Test environment

2.1 Execution environment

SSCRAP[1] is an environment designed to efficiently implement coarse grained algorithms regardless of whether or not the target architecture is shared memory or distributed. It allows an algorithm design according to the various coarse grained execution models while simplifying their implementation.

The coarse grained models provide a design framework with the aim to maximize the local computation and to minimize the global operations [2]. The most known coarse grained models are BSP (Bulk Synchronous Parallel model) [3], LogP (Latency overhead gap Processors number) [4], CGM (Coarse Grained Multicomputer) [2] and PRO (Parallel Resource-Optimal computation) [5]. PRO, BSP and CGM are defined as the union of three sub-models: an abstract machine model, an execution model and a complexity or cost model. Both, their architectural and execution models are similar. A CGM and BSP computer is a collection of processor/memory modules connected by a router that can deliver messages in a point-to-point fashion between the processors. The execution model describes the coarse grained algorithm as a sequence of *supersteps*. In each *superstep*, processors first perform computation on local data and then communicate to exchange the data required for the next *superstep*. *Supersteps* are separated by synchronization which can be invoked explicitly using barriers (BSP) or implicitly during reception (CGM).

In fact, a theoretical analysis shows that when we impose some reasonable assumptions $p \leq \sqrt{n}$ (for p the number of processors and n the size of the data) and by limiting the amount of supersteps to a "reasonable" amount the performance analysis of algorithms should become much simpler: the effect of the latency of the interconnection network is neglectable compared to bandwidth restrictions. For the design of real world clusters the impact of this observation can be crucial since high performance latency is much more expensive than bandwidth. But the usefulness strongly depends on the possibility to implement these concepts.

[1] Soft Synchronized Computing in Rounds for Adequate Parallelization, see [1]

For the SSCRAP design, the main goals were portability, efficiency and extensibility. The most difficult task was to balance between efficiency and portability. Indeed, to be portable, a program must consider a generic machine model disregarding the specificity of the physical architecture. However to be efficient, a program must get the full benefit from the available resources which differ enormously from one architecture to another. To ensure this task and the complete transparency for the user, we introduce an abstract communication layer between the "user" interface and the target platform.

Currently, SSCRAP is designed for Unix like platforms and interfaced with the three main types of parallel architectures: distributed memory (as e.g. cluster of PC or workstations), shared memory and hierarchical architecture (as e.g. cluster of SMP). By taking the differences between these architectural types into account, we implemented three different versions of the communication layer. The first one called SHM and based on *pthreads* is designed for a shared memory architectures. The second which was developed for the distributed memory architecture is based on MPI. Depending on target platforms, SSCRAP allows the choice between available MPI implementations. Thus, on the Origin2000 for example, SSCRAP can be interfaced with the LAM, MPICH or SGIMPI libraries. For the hierarchical architectures, the last added interface is developed upon PM^2 but because of installation problems we were not able to include test results in this paper.

2.2 Algorithms

For our tests, we consider three typical problems: *List Ranking*, *sorting* and *matrix multiplication*. These represent three different application types: applications with regular data (*sorting*), applications with irregular data (*List Ranking*) and applications with expensive local computation (*matrix multiplication*).

The *List Ranking* has a chained list of nodes as input. Each node knows its successor node as well as the distance which separates these two nodes. Solving the *List Ranking* problems consists in computing for each node the distance which separates it from the last node in the list. In contrast to the known theoretical complexity this problem is notoriously difficult to implement with acceptable speedups on few processors, see e.g. [6].

Cluster	Network	Lib	List Ranking	Sorting	Matrix Multi.
			Number Of SSCRAP Processes		
ICluster	Ethernet	LAM	1,2,..32	1,2,..10	NA
Albus	Ethernet	Mpich	1,2,..7	1,2,..7	1,2,..7
Albus	Ethernet	Mpich (+SMP)	8,9..14	8,9..14	8,9..14
Albus	Myrinet	Mpich	1,2,..6	1,2,..6	1,2,..6
Albus	Myrinet	Mpich (+SMP)	7,8..12	7,8..12	7,8..12
			Input Range (number of elements)		
ICluster	Ethernet		5.0E+6..2.0E+8	1.0E+7..1.0E+8	NA
Albus	Ethernet/Myrinet		1.0E+7..1.2E+8	1.0E+6..2.0E+8	1.6E+5..1.6E+7

Table 1: Overview over the set of experiments

For tests, we implemented upon SSCRAP the parallel *List Ranking* algorithm proposed in [7], the *Quick Sort* for the sequential sorting, the BSP parallel sorting algorithm based on *Over Sampling* proposed in [8] and a distributed algorithm for matrix multiplication base on cyclic circulation of column bloc.

2.3 Platforms

For the tests, we consider two different types of cluster. Both of these clusters use Linux 2.4.2 as their operating system. The first one is large PC cluster (Icluster at Grenoble) [9] with about 200 nodes. The nodes are fairly distributed among five 100 Mb Ethernet branches which are interconnected by five 1 Gb switches mesh. Each node consists of a PC desktop powered by a 733 MHz Coppermine INTEL Pentium III processor with 256 MB SDRAM PC100 local memory at 800 MB/s bandwidth and 10 ns latency.

The second architecture ("Albus" at Nancy) is a cluster composed of 8 bi-AMD Athlon MP 1500+(1333 MHz) SMP nodes. Every of these nodes has 1.0 GB 2100 DDR-SDRAM memory providing 2,1 Gb/s bandwidth at 6 ns latency. For the interconnection, each node is equipped with two different interfaces: Ethernet 100 Mb and Myrinet 2000 M3F. Both of these interfaces ensure interconnection respectively through a switched Ethernet network and a switched optical Myrinet network. The Myrinet card installed on 64b PCI port at 66 MHz provides 528 MB/s (half-duplex) at 9 μs latency on DMA (Direct Memory Access).

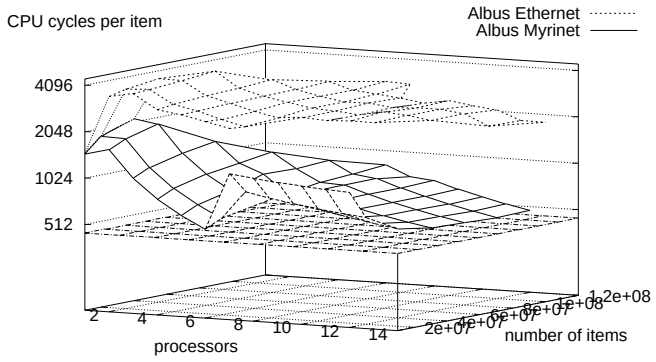
2.4 Test sets

Table 1 shows the different number of SSCRAP processes used during the different tests and the ranges of input data size that were considered. For all the tests, the SSCRAP process and input data are fairly distributed among the node. Thus, for 14 SSCRAP process on Ethernet tests, we have 2 process per node (one per processor) and $N/14$ elements per process (where N is the total input size).

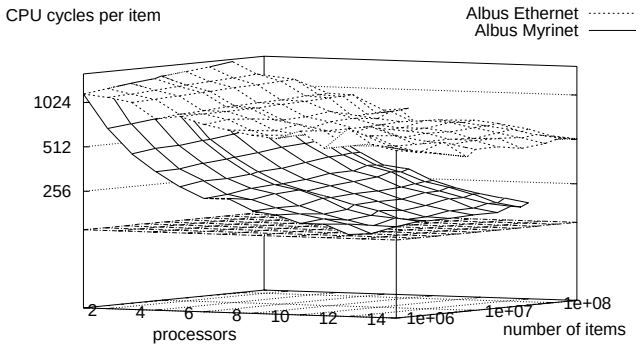
3 Results

Figure 1 shows execution times of *List Ranking* algorithm using both of Myrinet and Ethernet network interfaces on Albus. To be less dependent on the particular architecture and to ease the comparison to the sequential setting, the provided results are given in the number of clock cycles per list element in logarithmic scale. We notice that the Myrinet version is more efficient than the Ethernet one and results shows a typical coarse grained behavior: we obtain better performance if the problem input size is significant enough. We see also that we achieve a good scalability: the algorithm scales up to efficiently use the available memory on all machines.

Figure 2 corresponds to the sorting algorithm results on Albus with Myrinet and Ethernet Interfaces. We notice that, the sorting and *List Ranking* algorithms have the same behavior in Myrinet/Ethernet speed-up difference and scalability. In accordance with the theoretical lower bound for sorting, the per element Myrinet sorting cost in-



(a) List Ranking Myrinet/Ethernet on Albus



(b) Sorting Myrinet/Ethernet on Albus

Figure 1: Myrinet versus Ethernet on Albus

creases slightly with the total input size: the per-processor work is dominated by the local sorting cost which has an $O(\log(N/p))$ cost per element. As the Figures 2, 1 are slightly overloaded, the next figures will zoom the focus on some particular input sizes for a more detailed analysis.

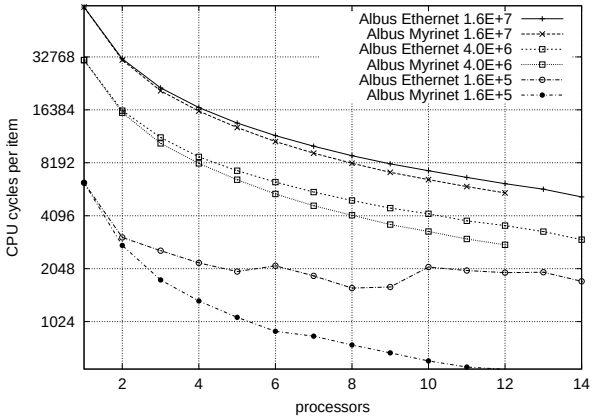


Figure 2: Matrix Multiplication Myrinet/Ethernet on Albus

Figure 3 represents the matrix multiplication results for Myrinet and Ethernet network interfaces on Albus. The input size corresponds to the number of element of the result matrix. Considering only three input sizes, we can easily notice that the larger input size, the smaller the gap between Myrinet and Ethernet results. As the overall local computing cost can be expressed as $O(N^{3/2})$, the computation time is $O(N^{3/2}/p)$ (where p is the number of process) and the overall communication cost as $O(N)$, the execution time is mainly dominated by computation. That explains the similarity between Myrinet and Ethernet results for large input.

Figure 4 corresponds to the sorting algorithm results on Albus and Icluster for 10 and 40 million doubles. Considering the sequential execution, we notice that results on ICluster (Ethernet) are better than both Myrinet and Ethernet Albus results. This looks disappointing but is simply due to the choice of expressing a *relative* efficiency compared to processor frequency: in sequential execution Albus' processors are starving whereas the ICluster processors are occupied to a large extent. That can be explained by the difference in ratio between CPU frequency and Memory Bandwidth on the two different types of nodes. In fact, in Albus, the CPU frequency is as high such that the available memory bandwidth is not large enough to satisfy the memory access requests.

However, as the data are distributed among the proces-

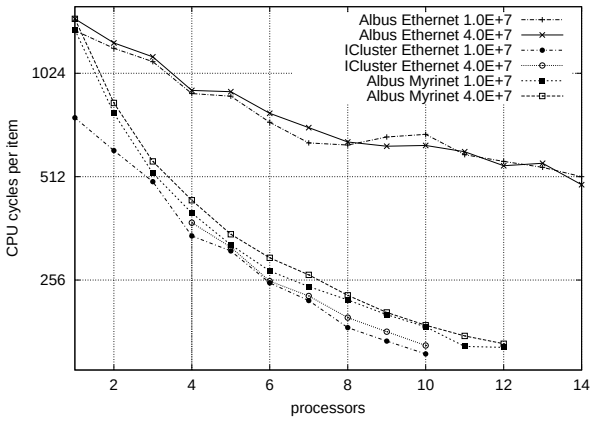


Figure 3: Sorting Albus (Myrinet/Ethernet) ICluster(Ethernet)

sors, in parallel execution the Albus/Myrinet and ICluster/Ethernet results are noticeably similar. This mean that the processors in Albus and ICluster have the same load. For Ethernet tests on Albus, the network bandwidth clearly represents a bottleneck.

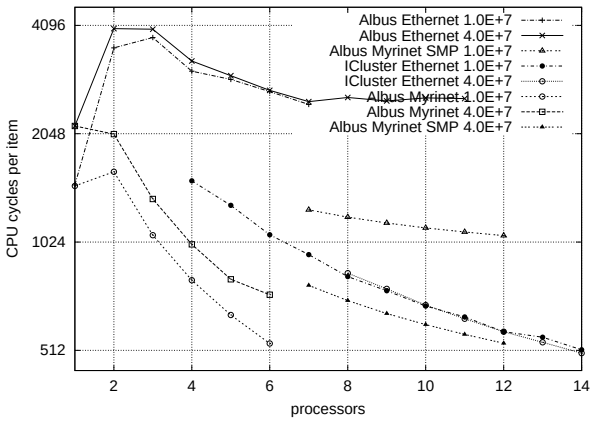


Figure 4: List Ranking Albus (Myrinet/Ethernet), ICluster (Ethernet)

Figure 5 shows the List Ranking results for Albus and Icluster. For a finer analyses we consider only two input sizes: 10 and 40 million list elements. For Albus, we distinguish the exclusive Myrinet execution ($p < 6$) and the combined Myrinet/SMP one ($6 < p < 12$). As List Ranking requires less memory access (per item) than Sorting, the Albus/Myrinet (without SMP) configuration is more efficient than the ICluster/Ethernet. On SMP execution and unlike the Mpich-Ethernet, the default Mpich-Myrinet interface doesn't take in account the process locality (in the same node) and thus it does not take advantages of sharing memory locally. That explain the per-item overhead observed on one node multiprocessing with Myrinet for 10 million element list. This per-item overhead is noticeably reduced when the input size is significant enough (40 million). For the Albus/Ethernet results, we obtained the same behavior as the Sorting results. Again, the Ethernet network bandwidth is the limitation for a good performance.

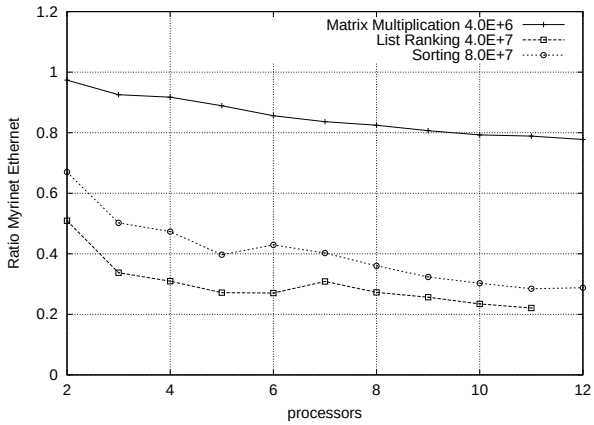


Figure 5: Relative Speedup between Myrinet and Ethernet

4 Conclusion

With Figure 6, we highlight the relative speedup between Myrinet and Ethernet (e.g. ratio CPU Cycle per item for Myrinet divided by CPU Cycle per item for Ethernet) taking in account one representative input size for each algorithm. Considering the ratio between global computing cost and global communication cost and classifying the algorithms in descending order we obtain: Matrix Multiplication ($O(\sqrt{n})$), Sorting ($O(\log(n))$) and List Ranking ($O(1)$). This order is clearly found in Figure 6 Matrix Multiplication (at most 20% performance gain), Sorting (70 %) and List Ranking (80 %). Figure 6 also demonstrates that from a practical point of view the use of recent and broader and broader network

interfaces is only interesting when the application requires a bandwidth that is comparable to the computation cost.

We also see that the theoretical observation about the dominance of network bandwidth over latency is effective with SSCRAP. Otherwise the number of supersteps (and thereby total number of messages) is would be more influencing the performance. The respective numbers are 3 supersteps for sorting, $\log p$ for list ranking and p for matrix multiplication, in particular a different ordering then observed for the efficiency.

Because of software limitations, we were not able to measure the possible performance gains provided by the locally shared memory in the bi-processor nodes. The only interface that was able to take a little bit advantage of this was MPICH on the 100 Mb/s Ethernet interface. But since here the main bottleneck is the bandwidth, this observation is more or less annecdotical.

References

[1] Mohamed Essaïdi, Isabelle Guérin Lassous, and Jens Gustedt. SSCRAP: An environment for coarse grained algorithms. In *Fourteenth IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2002)*, 2002.

[2] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel computational geometry for coarse grained multicomputers. *International Journal on Computational Geometry*, 6(3):379–400, 1996.

[3] Leslie G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103, 1990.

[4] David E. Culler et al. LogP: towards a realistic model of parallel computation. *ACM SIGPLAN Notices*, 28(7):1–12, 1993.

[5] Assefaw Hadish Gebermedhin et al. PRO: a model for parallel resource-optimal computation. In *16th Annual International Symposium on High Performance*, pages 106–113. IEEE, 2002.

[6] Jop F. Sibeyn. Ultimate Parallel List Ranking? In *Proceedings of the 6th Conference on High Performance Computing*, pages 191–201, 1999.

[7] Isabelle Guérin Lassous and Jens Gustedt. Portable List Ranking: an experimental study. *ACM Journal of Exprimental Algorithmics*, 22, 2002.

[8] Alexandros V. Gerbessiotis and Leslie G. Valiant. Direct Bulk-Synchronous Parallel Algorithms. *Journal of Parallel and Distributed Computing*, 22(2):251–267, 1994.

[9] `http://www-id.imag.fr/Grappes/icluster`.